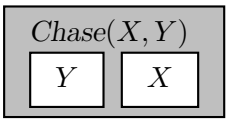
A CxG-inspired notation for glue

Avery A Andrews
Nov 2022, The Australian National University
draft; please do not criticize in print

LFG's glue semantics has a bit of a reputation for impenetrability, perhaps explicable by its original presentation in terms of linear logic deductions, which requires a a certain amount of background in relatively unfamiliar forms of logic.[1] Here I will present what I hope might be an easier to learn notation, based mathematically on the theory of proof-nets, expecially as developed by de Groote (1999) and Perrier (1999), but notationally a development of the notation of Construction Grammar, as presented by Kay & Fillmore (1999). Proof nets, as the name suggests, are a graphical representation for a restricted range of linear logic proofs, whose use I have discussed previously in Andrews (2010) and more demotically, in Andrews (2012), but I think the present proposal is an improvement in intelligibility over what is presented there. Construction grammar is an approach to the entire theory of grammar, whereas we will be using the notation only to represent the 'meaning-contributions' provided by lexical items from which the meaning of the sentence is put together compositionally.

1 Simple Boxes and Assembly

A basic notational idea of Construction Grammar is to represent 'constructions' (we won't worry about what these are to be taken to be) as boxes, containing sub-boxes into which other constructions are to be plugged (they are also related by inheritance networks, which are not used here). For reasons which will become apparent, we will want outer boxes to be shaded, and (the first level of) contained boxes to be unshaded, so the meaning-contribution for the verb *chase* might look like this, at a first approximation:

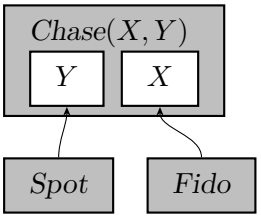
(1) 

The reversal of order of argument-variables between the meaning and the row of unshaded boxes will be explained eventually.

In a sentence such as *Fido chased Spot*, we would represent the meaning-contributions of *Fido* and *Spot* as shaded boxes containing no unshaded ones, and we could represent semantic assembly by drawing arrows from these shaded boxes to the unshaded ones in the meaning-contribution of the verb:

[1] 'Linear' and other forms of 'substructural' logic, in which various principles used in everyday reasoning about facts are dropped.

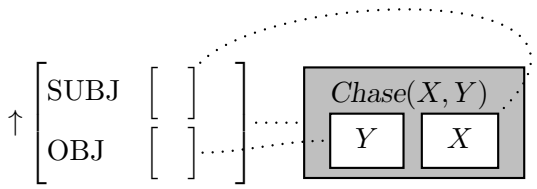
(2)



The intended meaning can then be produced by substitutions in a way that is hopefully obvious.

Before proceeding, it would be good to say something about how, in this approach, the syntax constrains the meaning, so as to exclude the other obvious way that the boxes could be hooked up, and the meaning that Spot chased Fido produced. The answer is that the boxes are introduced with connections to the syntactic structure; a meaning-contribution together with its syntactic specifications is called a 'meaning-constructor'. For present purposes, these can be represented graphically by dotted lines connecting them to pieces of f-structure, with an '↑' symbol attached to the f-structure correspondent of the lexical item (we will present the standard notation later). The entry for *chase* for example associated the X=Chaser role with the subject, and the Y=Chased role with the object:

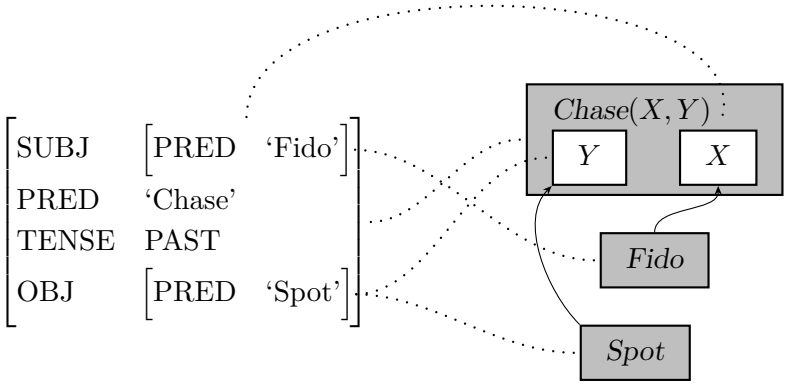
(3)



This says that the *Chase* meaning is associated with the f-structure of the verb node where *chase* is inserted, and that the X argument is found at the SUBJ of that f-structure, the Y at the OBJ of the f-structure.

Plugging a shaded box into an unshaded box then requires that both be associated with the same f-structure, so that the results for *Fido chased spot* are:

(4)



We here ignore the contribution of the past tense, to control clutter. You can see that the solid arrows that connect shaded to unshaded boxes require the dotted lines to connect both boxes to the same f-structure.

However the syntactic constraints are only part of the story; the plugging lines are also subject to various constraints on their own, that help define what coherent semantic assembly is. These rulwa come not from Construction Grammar, but from the theory proof nets, where they are called 'axiom links', because the represent the application of the deductive identity axiom $A \vdash A$. We will say a bit more about this later.

The inherently logical/semantic linking rules are:

(5) a. Every shaded box except one must be connected to one and only one unshaded box.

b. Every unshaded box must be connected to one and only one unshaded box.

c. If we imagine that the connections constitute sections of a directed path leading from shaded to unshaded, and that each unshaded box has an automatic (and not depicted here) connection to its immediately containing unshaded box then, from every shaded box there must be a path to the sole unconnected shaded box, which can be called the 'final output'.

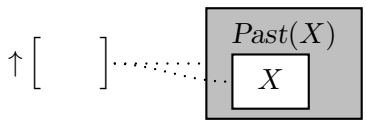
And one more rule with a bit of syntactic involvement:

(6) The final output must be associated (by dotted line) with the entire f-structre.

These rules block a lot of results we don't want, such as a semantic assembly consisting of just the verb without its arguments (assuming that in situations where arguments can be omitted from the overt form of an utterance, there are mechanisms to supply them).

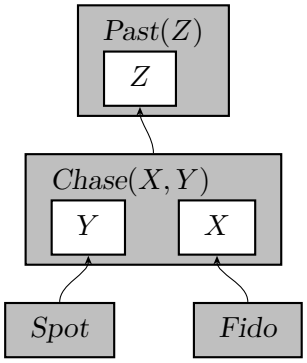
A more subtle situation that it blocks can be seen by considering the tense feature that we have ignored so far. A constructor for tense can be written like this:

(7)



If we add this to (4), we can get the following result, where we also introduce the refinement that when a meaning-contribution is introduced, you need to replace its variables with new ones that are not used elsewhere, in order to avoid various kinds of bad results:

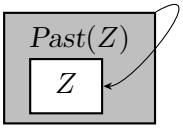
(8)



This will yield the result $Past(Chase(Fido, Spot))$.

The reader might have noticed that the rules constraining the plugging links have the effect of reinforcing the geometrical/topological intuitions that presumably underly Construction Grammar. For example if we are putting together a puzzle in which many pieces have holes into which other pieces are supposed to fit, it is quite natural that only one piece can fit into any hole, and that one piece cannot fill two holes at once, and that a piece cannot fill a hole in itself, as suggested by this rule-breaking example of mis-plugging:

(9)



I believe that it should be possible to formulate these intuitions rigorously in terms of topological notions, but have not worked out exactly how to do that.

A significant aspect of linear logic and glue semantics is 'types'. Formal semantics tends to assume a system of essentially different ontological types, starting with 'entity' (e) and 'proposition' (t).[2] So far we have only assumed one, but could add a conventional type system as additional information associated with the boxes, and requiring that for a positive box to be plugged into a negative one, their types as well as their f-structure locations must match. Types are an integral aspect of linear logic, so what we have done so far can be described as an ultra-rudimentary system with one basic type.

Indeed, what we have produced is the easy part of 'intuitionistic implicational linear logic' (IILL), which is implicit in the basic intuitions about ways of putting things together that were used by Fillmore and Kay in their development of construction grammar. It could be thought of as the logic of things, some of which have slots for inserting other things, to produce something different that uses up the items from which it was produced. For example, if you have a stool seat with three holes into which legs can be inserted, and also three legs, you can put them in to produce a stool, but you

[2]See Casadio (1988), for an interesting discussion of the background of this, and Partee (2006) and Partee & Borschev (2004) for further interesting discussion.

don't have the seat or the legs anymore (so you can't just use the original parts to make a new stool, winding up with two in all).

But now we move on to (a notational equivalent of) full IILL. For this, the easiest entry is perhaps quantifiers, especially the 'generalized quantifiers' of Barwise & Cooper (1981). The basic idea the is that quantifiers such as *every*, *some* and *most* involve comparisons between sets:

(10) a. Fido loves everybody

b. Fido loves somebody/some people

c. Fido loves most people

To evaluate (a), in a rather oversimplified way, we compare that set of people with the set set of entities that Fido loves, and accept the sentence is true if every item in the former set is in the latter, false otherwise. For (b), we want these sets to have some item in common, and for (c), we want the intersection of the set of people with the set of entities that Fido loves to be bigger than that of the set of people with the entities that Fido doesn't love. Part of the oversimplification here is that in normal usage, a sentence like this would be evaluated only on the basis of Fido's reactions to people he actually meets. We are not concerned with the entire population of human beings on Earth.

But how do we get a meaning like 'the set of entities that Fido loves' out of what we have so far? With our present ingredients, we can't, but need some additions, on the semantic side, 'lambdas'. These can be seen as a kind of mathematical version of relative clauses. We could paraphrase a discourse such as *I saw somebody. They were tall* as *The person (that/who) Fido loves is tall.* More generally, in syntax, a relative clause can be thought of as a sentence with a kind of variable/indeterminate element in it, along the lines of 'Fido loves (a person) x', We can use this clause as if it were an adjective by supposing that x refers to some specific individual, and checking whether the sentence come out true under that supposition. It can then be thought of as determining the set of people/ entities that Fido loves. The notation for this is:

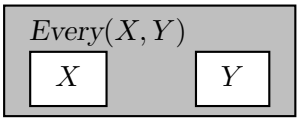
(11) $\lambda x.Loves(Fido, x)$

Once we have lambdas, it is highly desireable for the semantics to have something called 'types', to ensure useful formal properties, and make various things easier to understand. Here we would conventionally want the type of the variable x to be e (entity), and of $Loves(Fido, x)$ to be t (proposition), and the type of the entire expression of (11) will be $e \rightarrow t$, something into which you feed an e to get a t.[3] We might also consider subdividing the type e into 'people' and 'things', but considering the issues with higher

[3]In formal semantics, this would be most often written as $<e, t>$, but for the purposes

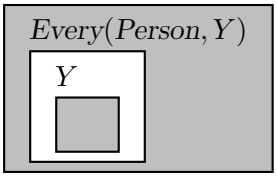
animals, the type concept is probably too rigid to be useful for this, and something involving the substantive meanings more appropriate.

Thinking of the meaning of *every* as something that relates two sets, a first stab at a meaning-contribution for *every* might be:

(12) 

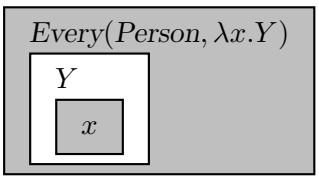
We could use types in the semantics to require that both X and Y are of type $e \rightarrow t$, and stipulate that the meaning of *person* is of that type, but how are we going to get anything like $\lambda x.Loves(Fido, x)$?

A way of thinking of it is that *Every* and similar meanings inject 'test entities' into a predicate (a proposition with a hole in it, suitable for an entity), to see whether they make the proposition true or not, so that they can then operate on the sets of entities such that the proposition is true. This suggests (at least if you already know about de Groote's and Perrier's work on proof nets) that there could be a shaded box inside the negative boxes of (12), which get linked to an unshaded box (essentially hole) in something of type $e \rightarrow t$. In the specific case of *everybody*, we can suppose that its meaning already puts *Person*, of type $e \rightarrow t$, into the X position, so the meaning contribution we get looks almost like this:

(13) 

But what do we put into the solid box, and how do we get a lambda-expression?

The answer will be that we put a lower-case variable into the shaded box, and a lamba-binder with it around the upper case variable of the unshaded box in the semantic formula, so that the finished product is:

(14) 

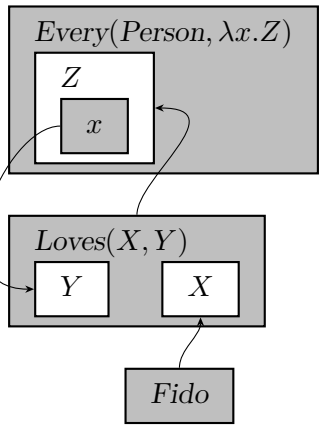
One question is what is the difference between the upper case and lower case variables, and the answer is that the former are part of the assembly

of linear logic and glue semantics, an arrow notation is normally employed, often the rather intimidating $\multimap$ symbol rather than $\rightarrow$, but I'll use the latter here. The reason for this is a relationship to implication in logic which will be discussed later.

mechanism, and can be understood in terms of simple substitution (if we remember to use different ones in each meaning-contribution!), while the latter are part of the meaning language, and get interpreted by the significantly more complex mechanisms of lambda-calculus. But until we look at that a bit harder, we can gloss $\lambda x.Y$ as 'x such that Y us true'. Another issue is the difference between what appears in the unshaded box and what appears in the meaning formula. For perspicacity, we want the 'initial meaning' of the unshaded box to be whatever gets piped in from the shaded box that links to it, but then we need to add the lambda-binding, so putting that into the meaning formula for the entire shaded box seems like a reasonable thing to do. This is dealt with more systematically a later section, where a more formal approach is taken, and the relationship to standard proof-nets explained.

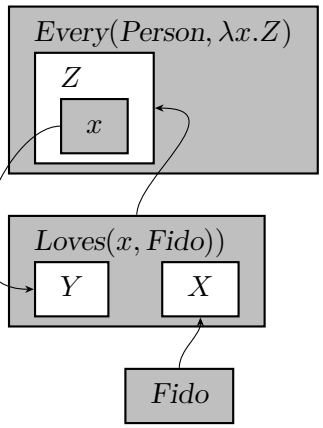
Now, the assembled meanings for *Fido loves everybody* can be represented as:

(15)

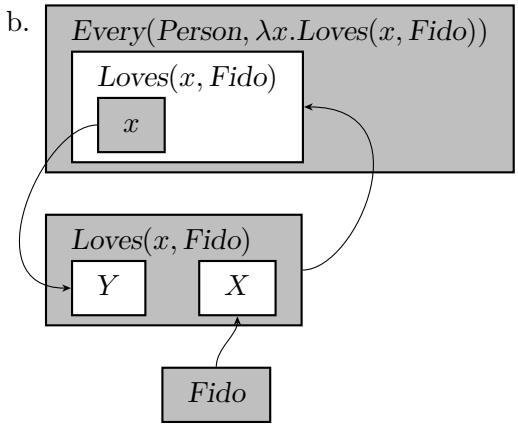


At this point, the propagation of semantic values through the network gets a bit complicated, so I will show it in two steps:

(16) a.



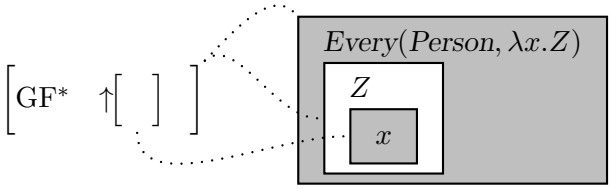
b.



So far, we have only considered the meaning-contribution for *every(body)*, but not the constructor, which has to provide information about how this connects to the syntax.

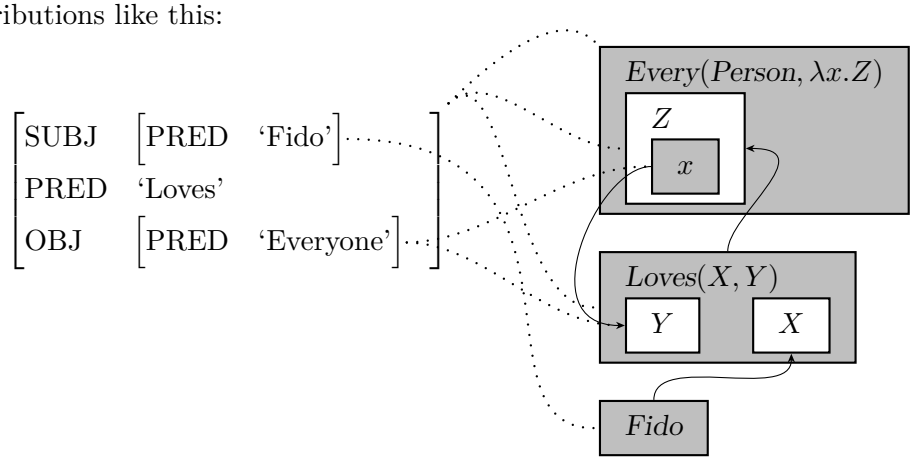
The classic problem is that *every(body)* is stuck inside a noun-phrase, so how does it get to apply to an entire sentence? There is more than one possible answer to this question, but here I will use the one proposed in Andrews (2010), that reference to higher structures in involved. The relevant technical LFG concept is 'inside-out functional uncertainty' (iofu). This is here depicted by putting the ↑ on an inner f-structure, corresponding to the quantified pronominal NP:

(17)



We can depict the connections between the f-structure and the semantic contributions like this:

(18)



The ↑ in (17) means that this is the f-structure of the NP the quantifier is introduced in, so connecting the inner shaded x box to it means that

the variable x is associated with that syntactic position, which means it is understood as the object of *loves*. Then the GF*, meaning 'a sequence of any number of grammatical functions, it doesn't matter what they are' is interpreted 'inside out' (or, 'upside down') to identify some containing structure, in this case the immediately containing clause.[4]

This extension to the notation requires a further addition to the linking rules:

(19) The path from a contained shaded box to the final output of the hookup (multiple contributions connected by axiom links) must go via the unshaded box immediately containing the shaded one (recall that these paths include implicit links from an unshaded box to their immediately containing shaded one).

The effect of this condition that a variable introduced by a contained shaded box cannot wind up outside of the scope of a lambda.

2 Multiple shaded contained boxes? application to an idiom

Quantifiers are a very traditional and perhaps to many people not very exciting application of these techniques, and they furthermore do not raise the issue of whether an unshaded box should be allowed to contain multiple shaded ones. Here I will take a look at idioms such as these:

(20) I read the crap/shit/hell out of that book

considered in a facebook posting by Omer Preminger.[5]

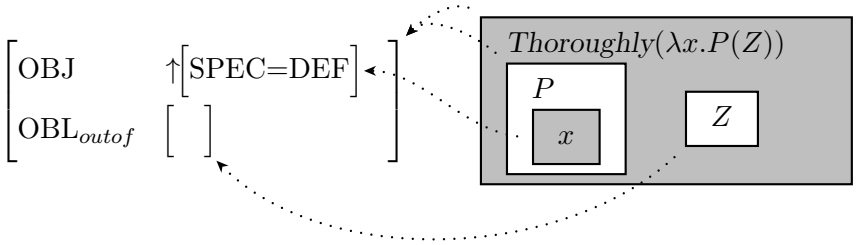
Roughly, what seems to be going on is that the normal direct object of a verb is displaced into an *out of* object position, with one of a variety of emotionally expressive nouns taking the object position (subject to numerous restrictions discussed in the facebook thread). The construction seems to submit, at least to a first approximation, to a treatment based on Asudeh, Dalrymple & Toivonen (2013), which utilizes a capacity that glue semantics has of appearing to rewrite grammatical relations.

In the facebook discussion, it was supposed that the construction effectively wrapped the verb meaning in an adverbial meaning similar to *extremely thoroughly*, and I think it is worth presenting an analysis along these lines first, to introduce a useful thing that glue can do, which is cause something introduced in a prepositional phrase (*out of*) to be interpreted as the direct object of the verb. This will only require one shaded inner box; after working through it, we will upgrade to two.

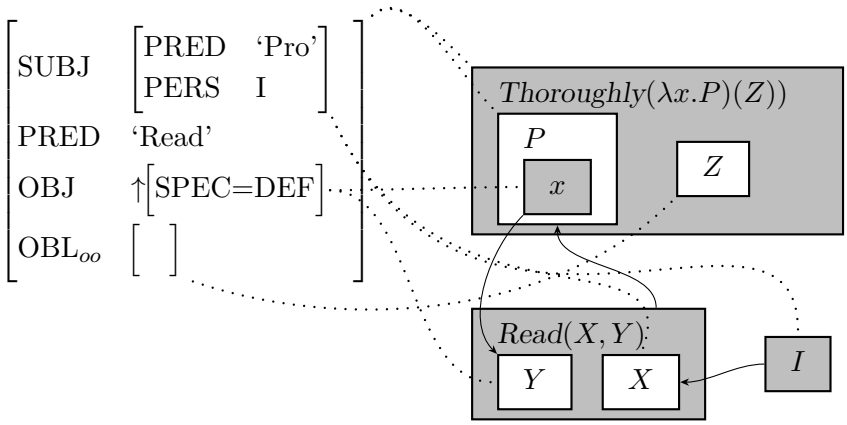
[4]For discussion of problems with this approach, see Gotham (2022).

[5]`https://www.facebook.com/omer.preminger/posts/1325774967772794`

Below is a constructor that does what is needed for the vfirest versio, associated with a noun such as *shit* that triggers this construction, where SPEC=DEF is a notation for the requirement that the specifier of the NP in this construction must be definite:[6]

(21)


The constructor supplies the variable x to the object position where *shit* with its idiomatic sense is inserted. This means that a transitive verb such as *Read* can form an open sentence such as $Read(I, x)$, assuming '*I*' as the subject. This then gets returned to the complex unshaded box, so that P is $Read(I, x)$, leading to $\lambda x.Read(I, x)$ being its ultimate contribution to the meaning. A partial assembly is:

(22)


Value propagation causes P to be $Read(I, x)$, producing

(23) $Thoroughly(\lambda x.Read(I, x))(Z))$

as the partially specified resulting meaning.

Whatever is sitting in the *out of*-object position (here abbreviated *oo* will supply the meaning that is to be substituted for Z, but either before or after dealing with that, we can apply the process of 'beta-reduction' to get rid of the lambda-expression, and substitute Z for x in P. Beta

[6] In standard LFG equational notation, this would be accomplished by a constraining equation on the relevent lexical item for *shit*.

reduction is a calculation process for simplifying expressions where a lamba-expression is applied to an argument by substituting the expression for the argument for every instance of the variable 'bound' by the lambda, subject to some safeguards. For a simple example from grade school arithmetic, the expression $(\lambda x.2 + x)(3)$ beta-reduces to 2+3.

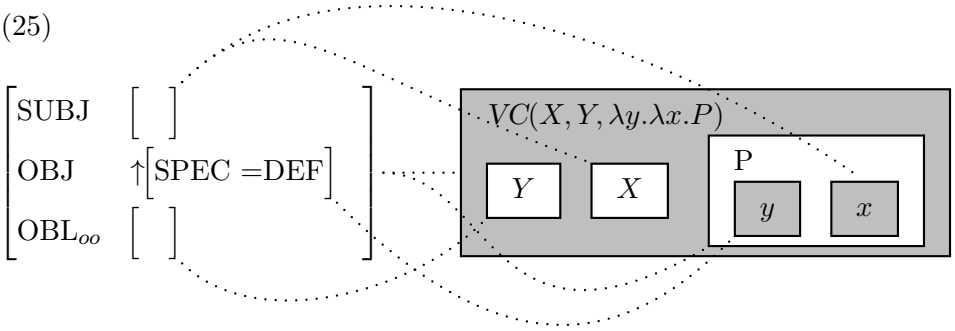
The safeguards are needed in order to assure that the results of beta-reduction mean the same thing as the expression reduced, and we can avoid the chore of learning about them by (a) using a different variable with each lambda (b) not using any variables that are not 'bound' by a lambda. But what is it for a variable to be bound by a lambda? To be clear enough about this, we need to distinguish between 'variables as kinds' and 'variables as occurrences'. This is very similar to distinguishing between 'book' as a composition with a text, author, publisher, etc, versus a physical object (*Cindy has written three books* vs *there are three books on the table* (possibly all the same volume in the 'kind' sense.)). So for example in $\lambda x.x * x$ we have three occurrences of the variable (as kind) x, the second and third bound by the lambda-expression involving the first. Now we can say that an occurrence of the variable x is bound in an expression E iff it occurs in a subexpression D of E (including the possibility that E is all of D of the form $\lambda x.P$). If a variable is not bound by something, it is free, so the bottom line is that we don't want any free variables in the (full) expressions we use, and we want all the lambda-binders to use different variables. For a gentle introduction to more on this subject, Coppock & Champollion (2022) is an excellent freely available source. Now we procede to the use of two shaded boxes in an unshaded one.

Although the semantic analysis we have so far this is not completely terrible, I think it is less illuminating than it could be, and, in particular, sheds no light on how expressions such as this might arise in the first place. What I suggest here draws heavily on the proposal of Egan (2004) that idioms are based on a pretence. Egan proposes this as a general analysis of all idioms, which I won't endorse here (and think there are probems with it), but it nevertheless is illluminating, especially perhaps for the more recent ones. The basic idea of this analysis is that the meaning of V the shit etc. out of expressions might be better described as:

(24) X Ved Y like Y hitting Y it so hard that shit/crap etc came out of it.

What want an analysis on these lines to do is grab the meanings of both the subject and the object, and, in effect, copy them (which the semantics can do, although glue/linear logic by itself cannot), with one pair going to the verb and the other going to the *like* clause. So were dealing with the meanings of two arguments, subject and object, rather than just one, as in the previous analysis. A tentative version of the meaning-constructor will be this, where VC is a convenient abbreviation for the meaning we propose:

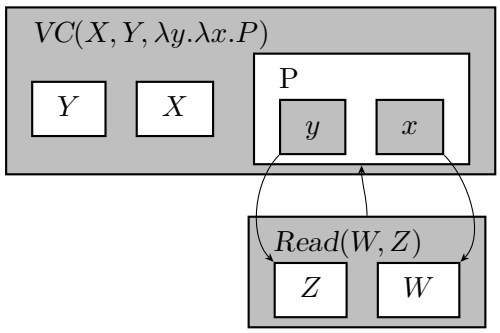
(25)



This works like the previous analysis, with the meaning changed to involve the subject as well as the object.

But there are some issues to discuss concerning the arrangement of the two shaded boxes in the unshaded P box. What order should they appear in? There is somewhere between very little and absolutely zero empirical content to this issue, but we get the least confusing notation for the commonest situation if we put them in the same relative order as the unshaded argument boxes of a verb that would provide a meaning for P. So adding a verb to (25) we get, (26), with clean hookup lines (but we omit the connections to the f-structure because they would add even more clutter):

(26)



Putting the lambda-bindings in the order used also assures that the so-packaged verb meaning wants its arguments supplied in the same order as the original verb. There is no empirical significance to this, but I think it should help ameliorate confusion in the formulas. So we can make a bit of progress on the proposed meaning of VC, presenting it as:

(27) $VC(X, Y, V) = V(Y)(X)$ like X hitting Y so hard that shit/crap etc came out of Y.

But what exactly is $V(Y)(X)$?

It is an example of what is usually called 'currying' a function of two arguments, even though it would perhaps better be called Schönfinkelization, because it was invented by the logician Moses Schönfinkel, although another logician H.B. Curry was able to write considerably more about (crediting

Schönfinkel, although people mostly applied Curry's name to it). It consists of presenting a function that takes two arguments as one that takes one argument to another function, that applies to the other argument, to produce the value. Explaining this more carefully requires taking a more 'hardcore' approach to glue, to which we procede now in the next section.

3 Getting more hardcore

We divide this section into two subsections, the first developing The present approach is sufficient for applications, but, ultimately, to do glue you need to know how it relates to the more widely encountered ones, and connecting it to conventional proof-nets is the first step (there is plenty of literature about the relations between proof-nets and deductions, the most widely used format in the literature, although not very perspicuous, it seems to me). I will approach this in two stages, first, aspects of formal semantics which people familiar with formal semantics can skip, second, things more specific to proof-nets.

3.1 Formal Semantics

The first step is to continue the discussion at the end of the previous section. This requires us to say more about types. So far we have briefly discussed types such as $e{\rightarrow}t$; we now want to get a bit more general and therefore formal about this. $\rightarrow$ is what is called a 'type constructor'; it constructs new types from old ones. In the formal definition of its use, parentheses are put around the constructed type, so what we have been writing as $e{\rightarrow}t$ is actually $(e{\rightarrow}t)$. This allows us to distinguish between $(e{\rightarrow}t){\rightarrow}t)$ and $(e{\rightarrow}(e{\rightarrow}t))$, both types we have encountered, but writing so many parentheses can get tedious, so there is a convention of omitting ones that are final, including outermost, leading to $(e{\rightarrow}t){\rightarrow}t$ and $e{\rightarrow}e{\rightarrow}t$ for the above two, and $e{\rightarrow}e{\rightarrow}(e{\rightarrow}e{\rightarrow}t){\rightarrow}t$ for VC from the previous section.

The application of functions to arguments can be stated as the following general rule:

(28) If f is of type $A{\rightarrow}B$, and a of type A, then $f(a)$ is of type B.

And we have already discussed the notational convenyion of applying a function to the parenthesized item immediately to the right, and then the result of that to the next parenthesized item, so that if f is of type $A{\rightarrow}B{\rightarrow}C$, and formulas a, b, c of types A, B, C respectively, then $f(a)(b)$ is of type c.

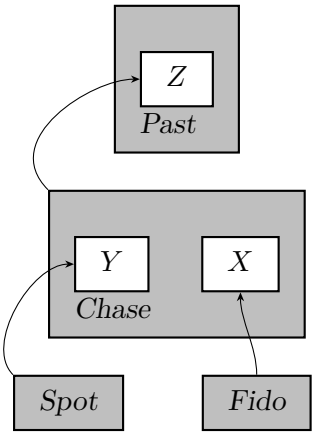
Since we have a type rule for function application, we should also want one for lambda-abstraction:

(29) If x is a variable of type a, and P is an expression of type b, then $\lambda x.P$ is an expression of type $a{\rightarrow}b$.

We are now ready to procede to the next step of explaining the relation between the box box representation of proof-nets to the conventional one, using trees sitting on their roots (rather than hung upside down from them, as usual in linguistics). We also change our treatment of semantic values, using lambda-calculus only rather than substitition for upper-case variables.

Moving on, the next step is to replace our use of upper case variables with linear order. In order to bring the treatment closer to that of proof nets, and elucidate the treatment of unshaded boxes containing shaded ones, boxes contain sub-boxes will have two meanings, not just one: an initial meaning, which can be sean as what the box starts out with, and a final meaning, derived by combination with the sub-boxes. We will write inital meanings at the bottom of boxes, final meanings at the top. But for boxes that contain no inner boxes, the initial and final meanings are the same, so we write them just once. The results for *Fido chased Spot* will then look like this, before any meaning-propagation:

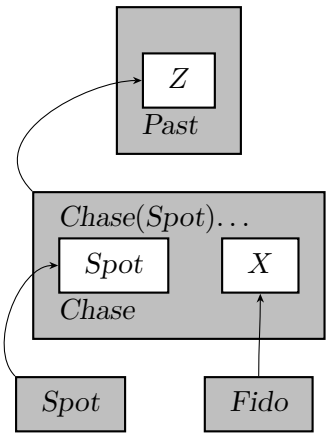
(30)



For a shaded box containing unshaded ones, the final meaning is produced by starting with the initial meaning, and then applying each argument, in the order of their unshaded boxes.

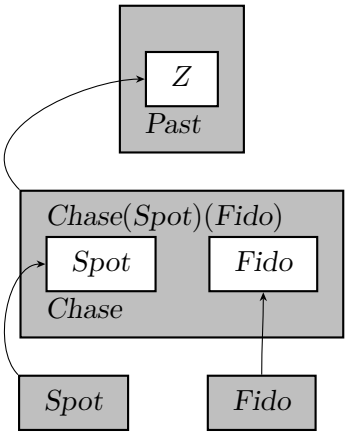
A first step would look like this:

(31)



A second, like this:

(32)



The last step, not written out in a diagram, produces the final result *Past*(*Chase*(*Spot*)(*Fido*)) as the final meaning of the unlinked box and therefore the meaning of the assembly. It can be proved that this bottom up method will always work, if the network obeys the rules governing linking.

3.2 Linear Logic, Proofs and Proof Nets

Glue semantics arose as a development from the discovery that the logic of implication has the same structure as the lambda-calculus, which is called the 'Curry-Howard Isomorphism' (CHI). We have already intimated this in the discussion of the meaning of $\rightarrow$ along the lines of 'if you give me something of type A, you will get something of type B. This corresponds to the logical rule of 'Modus Ponens', which says that if you have the propositions A and $A \rightarrow B$ (reading $\rightarrow$ is designating some kind of *if*; the philosophy of *if* is very complicated), then you also have B. On the other hand lambda-abstraction corresponds to to the rule of 'Hypothetical Deduction': if, from A (and maybe some other assumptions), you can conclude B, then you can conclude $A \rightarrow B$ (from those other assumptions, if any). What really gets the CHI off the ground is the fact that the lambda-calculation of beta-reduction also corresponds to a way of removing meaningless detours from proofs, and there is another one called eta-reduction, to be briefly mentioned later, which also does. For these reasons, $\rightarrow$ is often called the implicational type constructor, with the item to its left called its antecedent, the thing to its right its consequent. We will be using this terminology soon.

But the logic we are using is not ordinary logic as taught in beginning logic courses, but 'substructural' logic, in which the use of premises is restricted in various ways, in this particular logic, by the restriction that every premise must be used once, and once only. When presented as a restriction on reasoning about facts, this tends to strike people as insane, but it makes

sense for reasoning about resources that can be used up,[7] as we discussed in connection with stool seat and legs: we can think of the seat as an item of the type $leg{\rightarrow}leg{\rightarrow}leg{\rightarrow}stool$, and then deduce:

(33) $leg{\rightarrow}leg{\rightarrow}leg{\rightarrow}stool \vdash stool$

where $\vdash$ (read 'turnstyle') is used to separate premises on the right from conlusions on the left. In regular logic, it makes no sense to supply a premise multiple times, but in both linear logic and use of physical resources, you might need more than one copy of something, and if there is more than is needed, there will be some left over. So with 4 *leg* premises, the conclusion will be $stool, leg$.

For glue, we have been and will continue to operate under the assumption that we want a final output of type t, but this is unnecessarily restrictive; there are various other possibilities that might be suitable to the environment. For example, questions are sometimes analysed as being of type $e{\rightarrow}t$; a good answer is information that identifies an e to which this implication can be applied, so as to produce more information than present in the presuppositions to the question.[8] Similarly, if a group of people are looting abandoned structures, a collection of returns of type $e{\rightarrow}t$ (predicate nominals) might be appropriate: 'a crowbar! a sledgehammer!, pliers!!!'. How the environment determines what kind of final results are suitable would be an aspect of formalized or semi-formalized pragmatics.

3.3 Conventional Proof Nets

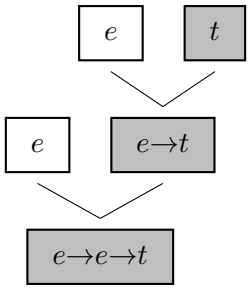
Now we explain the relationship between the box notation and conventional proof nets, which use trees, but sitting rightside up on their roots, rather than hung upside-down in the sadistic manner of linguistics. These trees are thought of as being a syntactic analysis of their type, which can be represented in various ways, but here I'll use shaded and unshaded boxes with various kinds of content, starting with just the types:[9]

[7] Even if somebody is reasoning about facts, somebody very interested in the structure of arguments might be interested in how often each premise is used, which basically amounts to allowing each (copy of a) premise to be used only once, and all premises to be so used.

[8] For example, the question *What will we have for dinner* presupposes that we will have food, rather than, say, dirt, so *food* is a bad answer, while *roast spice chicken* is a good one (at least if we have or can get the ingredients).

[9] This is pretty close to the representation of de Groote (1999) and much related work; that of Perrier (1999), who developed the interpretation methods, is substantially different in appearance

(34)

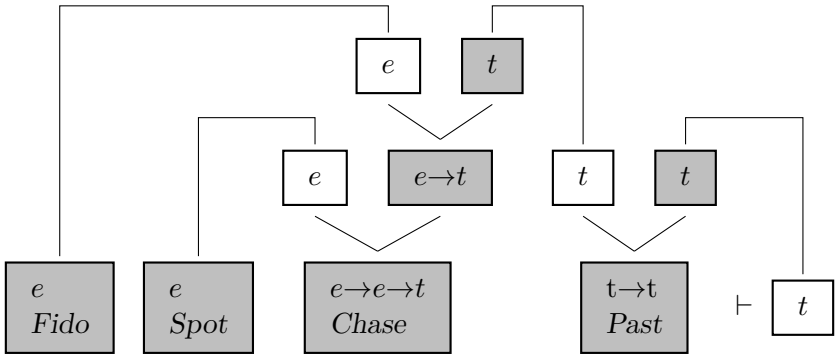


It should be evident that a shaded box could be regarded as a right branching 'spine' of shaded nodes, one for each unshaded box, which provides the left branch of the node, the antecedent of an implication, with the right being the consequent. And we also want meanings, which we will write underneath the types; examples to come soon.

The proof-net format is also different in another way; it is presented as a deductive argument with the premises to the left of a $\vdash$ symbol, and to the right of this we put a single conclusion, which will be an unshaded box. Shading is also construed as 'polarity', with values positive and negative, unfortunately, not with complete agreement about which nodes are positive and which negative. We will consider shaded boxes to be 'positive', on the basis of the intuition that they are supplying content, with the unshaded boxes negative, but the reverse is common (perhaps inspired by terminology from electricity, where Ben Franklin got it backwards?)

Axiom-links work like before, but as a consequence of the general setup, they go only from leaf-nodes, labelled with a basic type. They are represented by overbars (no arrows) connecting the nodes:

(35)

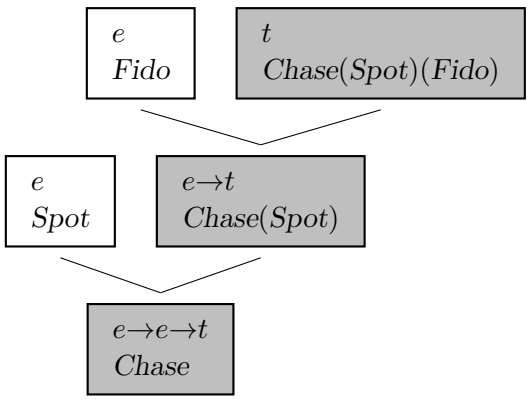


This style of diagram reveals the basis for the name a bit more clearly; they connect two instances of the same type, the shaded once constituing the premise, the unshaded one the conclusion, of an instance of the axiom $A \vdash A$. But the full story about how the diagrams represent deductive proofs is beyond the scope of this exposition.

Values are transmitted over the axiom links as before, and the rule for the conditionals is straightfoward: the value of the branch on the right (the 'consequent') is the value of the base applied to the value of the branch on the left (the 'antecedent').

Unfortunately, since the semantic values tend to get longer as the calculation procedes, a full calculation for the current example won't fit on the page, but we can do the result for the *Chase* tree, on the basis that the axiom links have propagated the values of the two arguments:

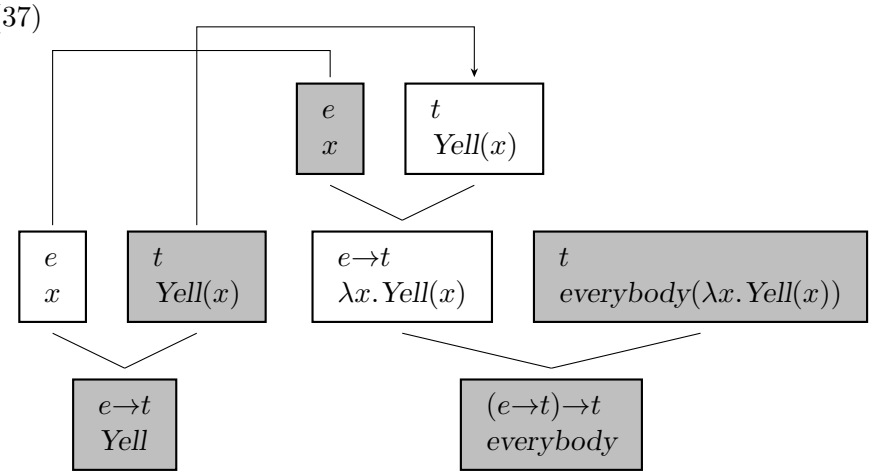
(36)



So we see that the initial meaning of a shaded box represents the meaning at the bottom of the spine, the final meaning the one at the tip.

Negative/shaded boxes and tree nodes have some similarity to negative/unshaded ones, but are also significantly different. A negative branching tree node has a positive antecedent (left branch) and negative consequent (right branch). A positive antecedent node also gets a meaning-variable assigned as its semantic value. In the simplest case, this node will be an atomic node, and the meaning will be propagated along an axiom link to somewhere, which, in accordance with the tree-based counterpart to (19), will have to return, (for linguistic applications, contained in other things), and is then wrapped in a lambda-binding using the variable of the antecedent, to become the meaning of the negative. We illustrate with the net for *everybody yells* (no tense):

(37)



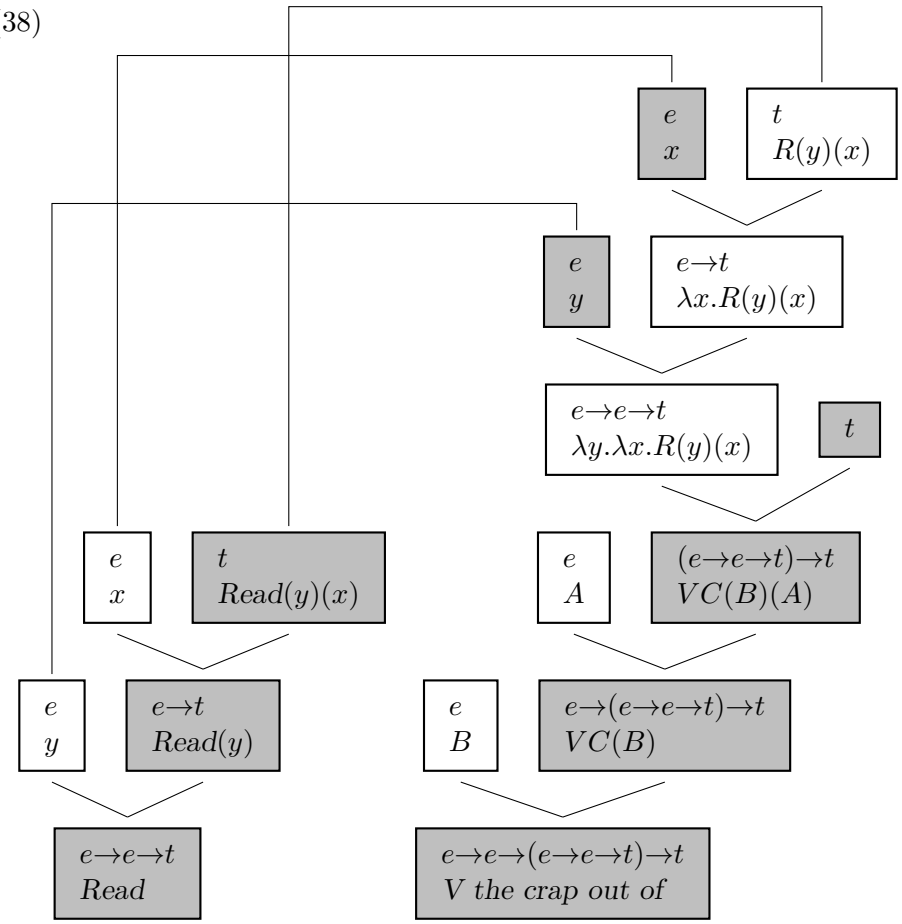
We leave out the turnstyle and connection to negative type t node to its right, both because it adds no new information, and won't fit horizontally on the page.

We can see here that in the simpler version of the box notation, we are putting the final meanings of the unshaded boxes into the semantic positions of the final meaning of the containing shaded box, telescoping a procedure rendered more explicitly in the tree notation.

The tree notation reveals two further possible complexities, and dictates what to do with them: either the antecedent of the consequent of a negative node may itself be complex. Complex antecedents would be needed to deal with Richard Montague's original analysis of 'intensional objects' such as *a unicorn* in *Mary seeks a unicorn*; the issue is discussed briefly for proof nets in Andrews (2010: 156-157). It is also clear how to handle it in the box notation; put a negative box inside a contained positive box. But I am not entirely convinced that we really need this for linguistic semantics.

But we have already used them for complex consequents, in our treatment of V the shit out of. Looking at the trees helps to clarify some issues involving the ordering of the variables in the contained shaded boxes and their lambdas. Here is a tree representation of the *VC* idiom composed with a verb:

(38)



No semantic value provided for the unconnected node, because it wouldn't fit. Note that the variable for the argument we want to supply first to the 'incorporated' verb (here *Read*) is the one lowest on the spine of negative nodes, last, highest.

We see that a 'semantically incorporated' transitive verb that is functioning as the type $e{\rightarrow}e{\rightarrow}t$ argument of something like *VC* should have its lambdas in the same linear order as the contained unshaded boxes, if the *VC* meaning is to apply arguments to them in the same order as they are applied to the verb. It is my suspicion that there is actually a problem in having to think about the linear order of occurrences in semantic representations, but actually resolving this problem is tricky.

We conclude by introducing a much more compact, but not so readable, notation which can also be used for meaning-constructors in their usual format in the LFG literature. In this format, a LFG meaning-constructor consists of a lambda-calculus semantic formula on the left, separated by a colon, which has a linear logic formula on its left. The constructors are introduced in the lexicon, where the atomic formulas in the linear logic formulas are normally represented as f-structure designators such as ($\uparrow$SUBJ)

or $\uparrow = \downarrow$. When an item is used in a structure, it is instantatiated and the label for its f-structure is used instead. But there is one more thing; glue has mostly assumed a 'semantic projection' coming of f- structure, called σ, but there was according to Andrews (2010) no convincing argumentation for it, so it has been omitted here. More recently, works such as Asudeh, Giorgolo & Toivonen (2014) and Findlay (2016) have developed more empirically founded uses for the semantic projection, but they are beyond the scope of this paper.

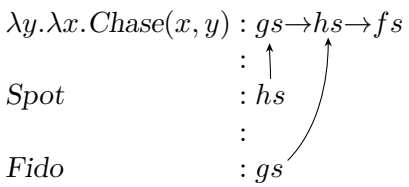
So for *chase* you would see something like (a) in a lexical entry, (b) in a structure:

(39) a. $\lambda y.\lambda x.Chase(x, y) : (\uparrow \text{OBJ})_\sigma \rightarrow (\uparrow \text{SUBJ})_\sigma \rightarrow \uparrow_\sigma$

b. $\lambda y.\lambda x.Chase(x, y) : gs \rightarrow hs \rightarrow gs$

The semantic types are usually omitted, as we have done with the box notation.

Now recall that in the regular proof-net notation, the trees are nothing but expanded presentations of the formulas, and there is no reason we can't just connect the atomic formulas directly to represent a proof, if we are confident in the readers' ability to understand it (and in our own not to make stupid mistakes). Then, the proof for the meaning of *Fido chased Spot* could be represeted like this:

(40)


This is pretty compact, but not perspicuous to the beginner.

The proof net notation, especially in this version, is on the whole more compact than the deductive formulations, but with one interesting exception. In deduction, you can apply a conditional with a complex antecedent of a type such as $(e \rightarrow t) \rightarrow t$, such as, say, *everybody* directly to a premise of the form $e \rightarrow t$, say, *yells*, to get the result *everybody*(*yells*). In proofnet glue, you can't do this, but have to work from the various atomic formulas (which are what is connected by axiom links) to get the result *everybody*(λx.*yells*(x)). The rule of eta-reduction, which we mentioned before, converts this latter into the former. One can imagine extensions to the theory of proof-nets that would deliver the more direct approach, but nobody has bothered to try to work them out rigorously, and indeed, Jay & Ghani (1994) have argued, on grounds I confess to not understanding, that the eta-expanded forms as produced by proof nets are better.[10]

[10]It is maybe worth commenting that formal semanticists seem to write things like

4 Limitations

This concludes the exposition of this diagramming technique, but I now want to say a bit about its limitations.

First, does not have a good representation of 'tensors', which have been proposed for various purposes, especially the treatment of anaphora and resumptive relative clauses in Asudeh (2004; 2012). Proof nets have be extended to handle these without issues, but the representation presented here struggles, and the geometric/topological intuition doesn't seem to be maintainable, as far as I have been able to tell. Another issue is monads, employed for various purposes in publications by Asudeh & Giorgolo (2016; 2020). I am not aware of any proof-net representation for monads at all, but I think it would be possible to do a preliminary, syntax-based analysis of the semantic composition of an utterance, using proof nets ignoring them (effectively, pretending that whatever monads were being used were the identity monad), and then putting in the monadic steps as needed into the proof represented by the proof net.

Neither of these are serious problems for this notation seen as an expository aid, because if you are ready to take on tensors, you can handle conventional proof nets, and if you are ready for monads, deductions will not be a problem either.

5 Further Reading

A wide-ranging but incomplete discussion of the background to glue semantics is Crouch & van Genabith (2000). A deeper (but for me relatively readable and illuminating) presentation of the basic ideas of proof equivalence, normalization, and the Curry-Howard Isomorphism, which are fundamental to the ideas behind glue semantics, is Girard, Lafont & Taylor (1989), while Troelstra (1992) is a standard introduction to linear logic, fairly accessible as such things go.

Dalrymple (2001) provides a thorough exposition of glue in the context of a comprehensive presentation of LFG, with analyses of various constructions, while Asudeh (2004; 2012) also provide thorough presentations with analysis of anaphora using tensors. Asudeh (2022) is a recent overview, including extensive citations to the literature.

$\lambda x.yells(x)$, rather than applying eta-reduction; I conjecture that the reason is that this gives enough information about the valence of the item, without requiring any specific decisions about what the basic types are, which is good, because that issue is not relevant to many discussions.

References

Andrews, Avery D. 2010. Propositional glue and the correspondence architecture of LFG. *Linguistics and Philosophy* 33. 141–170.

Andrews, Avery D. 2012. *Yet another attempt to explain glue*. `http://AveryAndrews.net/Papers` (Tutorials).

Asudeh, Ash. 2004. *Resumption as resource management*. Retrieved November 15, 2010, from `http://http-server.carleton.ca/~asudeh/`. Stanford CA: Stanford University. (Doctoral dissertation).

Asudeh, Ash. 2012. *The logic of pronominal resumption*. Oxford University Press.

Asudeh, Ash. 2022. Glue semantics. to appear in *Annual Review of Linguistics* `http://www.sas.rochester.edu/lin/sites/asudeh/pdf/Asudeh-AR-corrected.pdf`.

Asudeh, Ash, Mary Dalrymple & Ida Toivonen. 2013. Constructions with lexical integrity. *Journal of Language Modelling* 1. 1–51.

Asudeh, Ash & Gianluca Giorgolo. 2016. Perspectives. *Semantics & Pragmatics* 9. DOI: `http://dx.doi.org/10.3765/sp.9.21`.

Asudeh, Ash & Gianluca Giorgolo. 2020. *Enriched meanings: natural language semantics with category theory*. Oxford University Press.

Asudeh, Ash, Gianluca Giorgolo & Ida Toivonen. 2014. Meaning and valency. In *Proceedings of LFG14*, 68–88. CSLI Publications.

Barwise, Jon & Robin Cooper. 1981. Generalized quantifiers and natural language. *Linguistics and Philosophy* 4. 159–219.

Casadio, Claudia. 1988. Semantic categories and the development of categorial grammar. In Emmon Bach R.T. Oehrle & Deirdre Wheeler (eds.), *Categorial grammar and natural language semantics*, 95–123. Reidel.

Coppock, Elizabeth & Lucas Champollion. 2022. *Invitation to Formal Semantics*. `https://eecoppock.info/bootcamp/semantics-boot-camp.pdf`.

Crouch, Richard & Josef van Genabith. 2000. *Linear logic for linguists*. URL: `http://www.coli.uni-saarland.de/courses/logical-grammar/contents/crouch-genabith.pdf` (checked April 22 2013).

Dalrymple, Mary. 2001. *Lexical Functional Grammar*. Academic Press.

de Groote, Philippe. 1999. An algebraic correctness criterion for intuitionistic multiplicative proof-nets. *Theoretical Computer Science* 224. Retrieved 15 November, 2010, from `http://www.loria.fr/~degroote/bibliography.html`, 115–134.

Egan, Andy. 2004. *Pretense for the complete idiom*. Forthcoming in *Noûs*. URL: `http://www.sitemaker.umich.edu/egana/files/idiom.2006.11.09.pdf`.

Findlay, Jamie. 2016. Mapping theory without argument structure. *Journal of Language Modelling* 4. 293–338.

Girard, Jean-Yves, Yves Lafont & Paul Taylor. 1989. *Proofs and types*. Retrieved 15 November, 2010, from `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.85.5358&rep=rep1&type=pdf`. Cambridge: Cambridge University Press.

Gotham, Matthew. 2022. Approaches to scope islands in lfg+glue. In Miriam Butt & Tracy Holloway King (eds.), *Proceedings of lfg2021*. to appear. Stanford CA: CSLI Publications.

Jay, C. Barry & Neil Ghani. 1994. *The virtues of η-expanson*. URL: `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.39.613`.

Kay, Paul & Charles J. Fillmore. 1999. Grammatical constructions and linguistic generalizations: the what's X doing Y? construction. *Language* 75. 1–33.

Partee, Barbara H. 2006. Do we need two basic types. In Hans-Martin Gaertner et al. (eds.), *Puzzles for manfred krifka*. URL: `www.zas.gwz-berlin.de/40-60-puzzles-for-krifka/`. Berlin.

Partee, Barbara H. & Vladimir Borschev. 2004. Genitives, types, and sorts. In Ji-yung Kim, Yury A. Lander & Barbara H. Partee (eds.), *Possessives and beyond: semantics and syntax*, 29–43. URL: `http://people.umass.edu/partee/Research.htm`. Amherst MA: UMass GSLA.

Perrier, Guy. 1999. Labelled proof-nets for the syntax and semantics of natural languages. *L.G. of the IGPL* 7. 629–655.

Troelstra, A. S. 1992. *Lectures on linear logic*. Stanford CA: CSLI Publications.